Data Analysis and Machine Learning using Python

Lecture 3: Statistics, Fitting, Design strategies

March 23 2024

Menu for today

- Quick look at HW solutions
- Finish discussion of statistics
- Fitting discussion and example
- Design strategies
- Outlook on HW3

Some thoughts on program design

- We talked about Decomposition and Abstraction
    - For anything more than few dozen lines of code, it will pay to **think before coding** and to modularize/decompose code through functions (or objects)
- Although we are not developing a huge new application, the homework projects will be complex enough that some care is needed in thinking about the structure of the program before we get to work
- There are different approaches to software design - look at some of the different categories in general terms

Top-Down Design

- Complex problem is broken down at conceptual level into a collection of simpler/smaller problem problems
- Can iterate process by further breaking down these smaller problems into even simpler ones
- Continue until all problems have trivial or at least straightforward solutions
- Then successively combine the smaller, now solved, problems into a solution for the original problem

Design of top level

- At the top level, may have a simple input - process - output structure
- In designing top level(s), *assume* that all the lower level components needed to solve the problem exist, and that you just have to finish top-level algorithm using these components, i.e., employ the power of **abstraction and decomposition**

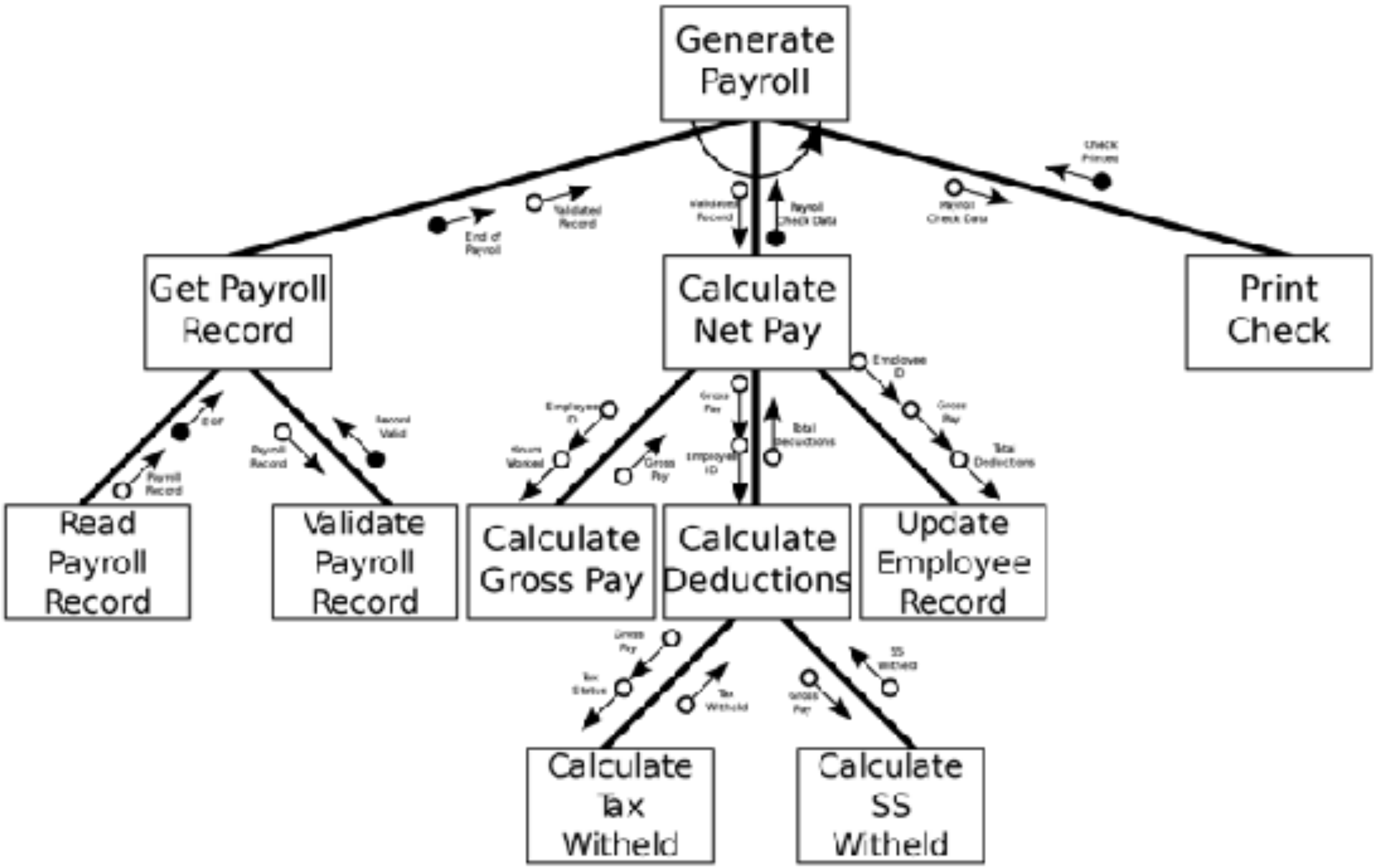
Interfaces and structure chart

- As the original problem is being decomposed into independent tasks, one defines names, input parameters, and expected return values of these smaller independent tasks (i.e., **functions)**
- Specification defines *interface* or *signature* for the smaller tasks

Interfaces and structure chart

- Having this information (the *interfaces*), allows us to work on each of these pieces independently.
- *Structure charts* that document the relationship between component at different design levels are an important tool
- e.g., line connecting two rectangles indicates that the one above uses the one below
- Arrows and annotations document interfaces between the components

Interfaces and structure chart



Interfaces and structure chart

- Interface documents/defines the relevant details of the lower level components at each level
- Defining functionality through interface, while ignoring “implementation details”: abstraction
- Systematically discover useful abstractions through top-level design process
- Follow by step-wise refinement turning abstract specification into concrete code conforming to the defined interface

Top-down design summary

1. Express the algorithm as a series of smaller problems.
2. Develop an interface for each of the small problems.
3. Detail the algorithm by expressing it in terms of its interfaces with the smaller problems.
4. Repeat the process for each smaller problem
5. Implement the actual algorithms for lower level components

How to make sure the whole thing works?

- Careful design of structure and interfaces doesn’t prevent mistakes at implementation
- Regardless of design process, implementation has to be done in small steps, with testing at each level of implementation
- **This is generally true of any complex project/activity - cannot manage/debug/test only at the highest level or with finished product**
- Need constant verification of pieces and interfaces as they are being implemented

Unit Testing

- **Unit testing** refers to systematically verifying the implementation of each modestly sized component,
  - Starting at the lowest/smallest component level
  - Testing as each component is completed
  - Testing when integrating into the overall program structure
- Proper functioning of each unit is a necessary condition for functioning of the whole.
- Not sufficient, as we also need to make sure interfaces are properly adhered to at each level

Unit Testing

- Execute and evaluate each unit for range of input parameters, in particular including extreme cases
- Deciding on the range of test cases requires careful thought
- Testing each function independently makes it easier to spot errors, and should then facilitate testing the entire program

Spiral development

- Top-down design is only one of many possible design approaches
- Alternative approach is **spiral development**
  - Start by solving a simplified version of the problem
  - Go through a implementation and testing cycle
  - Gradually add additional functionality in further cycles
  - Continue until the program matches the full specification

Spiral development

- Example: Upcoming sensor data homework
  - First prototype: read data and plot time series for one detector
  - Second cycle: add second detector
  - Third cycle: add time difference calculation and histogram for 10000 events
  - Fourth cycle: Speed up algorithm to allow processing of full data set in finite time
- The initial version is often a *prototype*
- Lessons from the prototype can be used to refine the overall design and structure
- Sometimes intermediate versions can be already deployed as development cycles continue

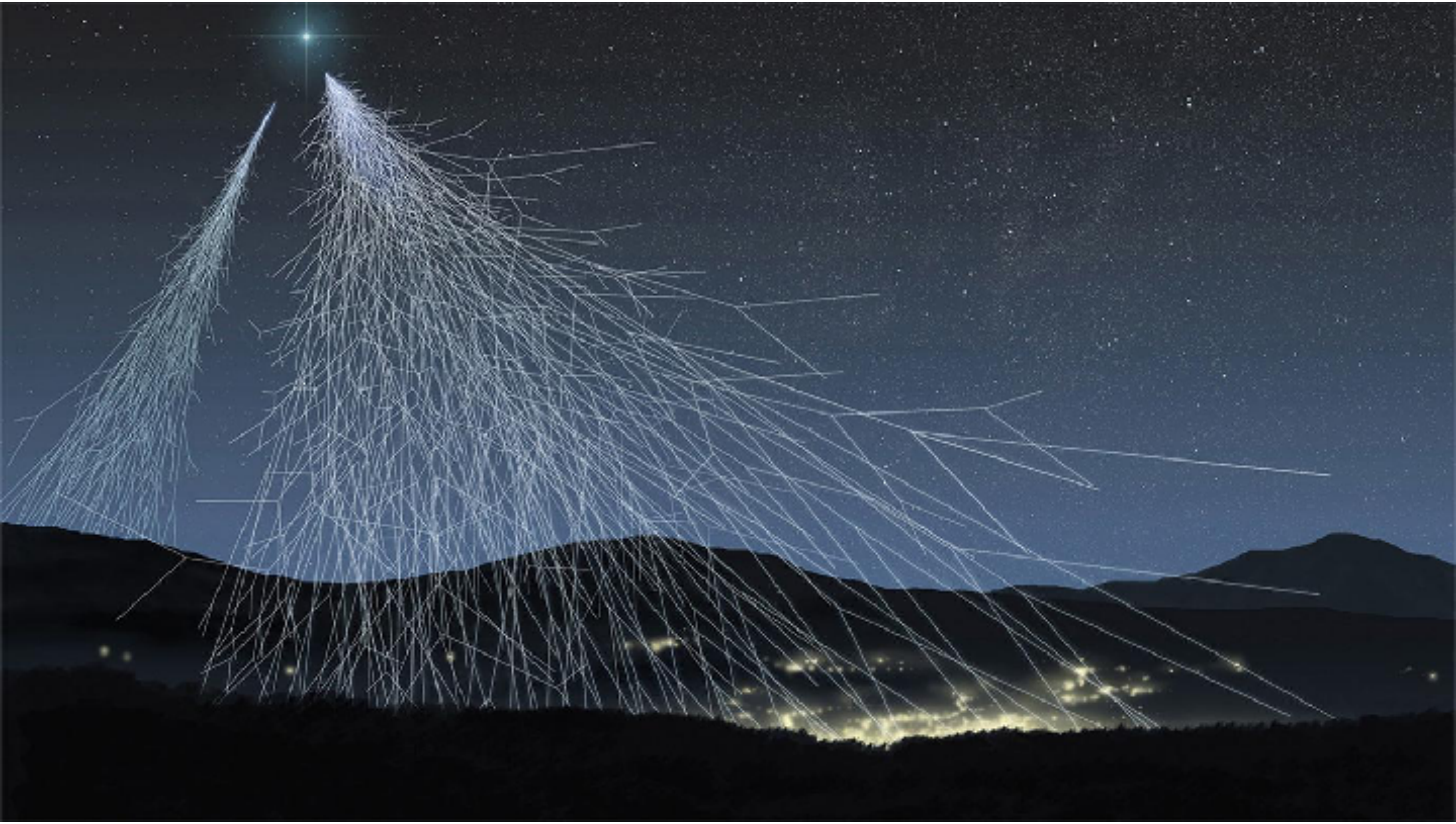
Spiral development

- Prototyping and spiral development can be particularly useful when solving unfamiliar problems using new, unfamiliar or unproven technology
- Design approaches are not exclusive - one can elements of top-down design with a spiral development/prototyping approach
- Find what works for you and your team - in the end, no process is a replacement for creativity and hard work
- But, expect that trying to write and test the whole thing at once, with testing after it's possibly done, will only work for the very simplest of problems!
- As always, practice makes perfect!

Cosmicwatch detectors

- For future in-class discussion and homework we will use Cosmicwatch detector data
- This gives real world examples of how to start from raw data to arrive at statistically valid conclusions based on various tools and concepts of data analysis
- This is not a physics course, so will not spend much time on the underlying physics

Cosmic rays



Cosmic watch detectors

Cosmic ray particle

When cosmic ray particle goes through detector, it leaves a signal

The time of the signal is recorded as the time stamp

But 90% of the signals recorded are electronic noise and radioactive background

How can extract the cosmic ray signal from the background?

- Data cleaning
- Multivariate analysis
- Correlation functions
- Machine learning classification
- Unsupervised learning?
- Deep learning?